UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
09/748,098	12/21/2000	Nicholas J. Kelsey	20880-05093	4360

758 7590 12/03/2008

FENWICK & WEST LLP
SILICON VALLEY CENTER
801 CALIFORNIA STREET
MOUNTAIN VIEW, CA 94041

EXAMINER	
HUISMAN, DAVID J	

ART UNIT	PAPER NUMBER
2183	

MAIL DATE	DELIVERY MODE
12/03/2008	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	09/748,098	KELSEY ET AL.	
	Examiner	Art Unit	
	DAVID J. HUISMAN	2183	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE *3* MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *17 September 2008*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-57* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-57* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *21 December 2000* is/are: a)☒ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____.
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. Claims 1-57 have been examined.

Papers Submitted

2. It is hereby acknowledged that the following papers have been received and placed of record in the file: Amendment as received on 9/17/2008.

Claim Objections

3. Claim 57 is objected to because the first paragraph is grammatically incorrect and applicant is further limiting thread selection hardware with a sequence of method steps. Appropriate correction is required.

Claim Rejections - 35 USC § 103

4. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

5. Claims 1, 29-33, and 42-47 are rejected under 35 U.S.C. 103(a) as being unpatentable over Gee et al., U.S. Patent No. 6,374,286 (herein referred to as Gee).

6. Referring to claim 1, Gee has taught a computer based system for switching between program contexts comprising:

a) a processor capable of having a first program thread and a second program thread in an execution pipeline having a thread selection hardware. See column 20, lines 42-52, and note that periodic threads exist. Also, see column 9, lines 36-38, and note that the system is pipelined.

b) a first set of data storage devices capable of storing a first thread state of said processor. See Fig.10 and column 19, line 48, to column 20, line 36. Essentially a separate thread control block (TCB) exists for each thread to hold information pertaining to that thread. Information is also stored in CPU registers.

c) a second set of data storage devices capable of storing a second thread state of said processor. See Fig.10 and column 19, line 48, to column 20, line 36. Essentially a separate thread control block (TCB) exists for each thread to hold information pertaining to that thread. Information is also stored in CPU registers.

d) Gee has not explicitly taught a hardware thread scheduler for identifying which of said program threads said processor executes and configurable to allocate available processing time of the processor among at least the first and second program threads by causing thread-switching at a fixed time according to a predetermined fixed schedule, said schedule specifying that the first thread should be allocated processing time every first number of cycles and that the second thread should be allocated processing time every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles. However, see Fig.19, column 20, lines 42-52, and column 32, lines 7-31, and note the "piano roll" scheduler for scheduling periodic interrupts. To summarize this scheduling concept, each column in Fig.19 corresponds to a thread assigned to a particular priority. For instance, column 3 represents a thread having a priority value of 3. And, column 29 represents a thread assigned a priority value of 29 (hereafter

"thread 29"). Each row in Fig.19 represents a chord, where the N^{th} chord is "accessed" or "played" at the N^{th} interrupt triggered by the expiration of a hardware timer. So, for example, if a periodic interrupt timer is set to expire every 100 ns, then the first time it expires at 100 ns, row 0 will be accessed. At 200 ns, row 1 will be accessed. At 300 ns, row 2 will be accessed, and so on. In each row, bits may be set to enable periodic interrupts. For instance, in Fig.19, the user has set up thread 29 as being allocated processing time every time the timer expires (due to the entire thread 29 column being filled with ones). If the interrupt timer is set at 1 second, then thread 29 executes at a rate of 1 Hz (i.e., it is allocated processing time once a second, which corresponds to a number of cycles depending on the speed of the machine). Although a specific combination of periodic threads has not been taught, one of ordinary skill in the art would have recognized that setting up a fixed schedule of periodic threads in Gee is well within Gee's capability. Certainly, Gee could not reasonably be expected to teach every possible combination of periodic threads, but Gee's intended functionality is apparent to one of ordinary skill in the art. Assuming a timer interrupt of 1 Hz (for simplicity) and a cleared piano roll table, thread 21 could be set up such that it executes every other chord (so instead of its bit being set in every row, it could be set in every other row (0, 2, 4, 6, etc.), thereby giving it an execution frequency of 0.5 Hz, or an allocation rate of once every 2 seconds). Similarly, thread 19 could be set up such that it executes every 4^{th} chord (so its bit will be set to '1' in rows 1, 5, 9, etc., thereby giving it an execution frequency of 0.25 Hz, or an allocation rate of once every 4 seconds). Finally, thread 17 could be set up such that it executes every 4^{th} other chord as well, but starting in a different cycle (so its bit will be set to '1' in rows 3, 7, 11, etc., thereby giving it an execution frequency of 0.25 Hz, or an allocation rate of once every 4 seconds). With the piano roll table set in this

manner, the execution of threads would be 21-19-21-17-21-19-21-17, etc. From this, it is clear that thread 21 is allocated processing time every X/4 cycles, where X is the number of cycles in 1 second. And, thread 19 is allocated processing time every X/2 cycles. Again, although this specific example was not taught by Gee, the functionality of Gee's piano table allows for such a configuration. To Gee, the specific examples were apparently not as important as setting forth the general concept behind the table. However, one would see the advantage of employing periodic threads to accomplish time-related tasks. For instance, in a real-time system, one might want to poll for a particular button input every 5 seconds and read a temperature sensor every 10 seconds. Gee's system allows for such flexibility. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to configure Gee's piano table such that that the first thread is allocated processing time every first number of cycles and that the second thread is allocated processing time every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles.

7. Referring to claim 29, Gee, as modified, has taught a system as described in claim 1, wherein said thread selection hardware in the pipelined processor switches between said first and second thread state after the end of the execution of a first program instruction in the first thread and before the beginning of the execution of a second program instruction. This is deemed inherent because thread "A" will execute for some amount of time and then a switch will occur to another thread. The switching marks the end of executing an instruction from thread "A" and the beginning of executing an instruction in thread "B". And, clearly, the system must be in the second state before it can begin executing the second thread.

8. Referring to claim 30, Gee, as modified, has taught a system as described in claim 1, wherein said processor is an embedded pipelined processor. See column 1, lines 22-27.

9. Referring to claim 31, Gee, as modified, has taught a system as described in claim 1, wherein said first state is the state of the processor during the execution of the first program thread. See column 20, lines 30-36.

10. Referring to claim 32, Gee, as modified, has taught a system as described in claim 1, wherein that said second state is the state of the processor during the execution of the second program thread. See column 20, lines 30-36.

11. Referring to claim 33, Gee, as modified, has taught a system as described in claim 1, wherein said processor switches between said first and second state by changing a state selection register. See Fig.10 and column 19, line 53, to column 20, line 36, and note that these registers are updated upon thread switches.

12. Referring to claim 42, Gee, as modified, has taught a system as described in claim 1, wherein said processor is capable of restoring said second state of said processor during execution of said first program thread. This is deemed inherent from column 20, lines 30-36, which states that registers store the current state of the currently active thread. Hence, at some point during execution of the first thread, a thread switch will occur, and the second state is restored to said registers. The switching to (restoring the state of) the second thread occurs during execution of the first thread.

13. Referring to claim 43, Gee, as modified, has taught a system as described in claim 1, wherein said processor is capable of storing said second thread state of said processor during execution of said first program thread. This is deemed inherent from column 20, lines 30-36,

which states that registers store the current state of the currently active thread. Hence, when a first thread is switched in and executed, the state of the second thread must be switched out and written to memory. Otherwise, the second state will be lost.

14. Referring to claim 44, Gee, as modified, has taught a system as described in claim 1, wherein said first set of storage devices comprises registers shared by a plurality of threads. See column 20, lines 30-36. All threads, at some point, have a currently executing state which is stored in registers.

15. Referring to claim 45, Gee, as modified, has taught a system as described in claim 1, wherein the predetermined fixed schedule is one of a fixed strict schedule, a semi-flexible strict schedule, and a loose strict schedule. Gee has taught at least a fixed strict schedule where threads are switched based on the data in Fig.19.

16. Referring to claim 46, Gee, as modified, has taught a computer based method for switching between program contexts in a multithreading pipelined processor (see column 9, lines 36-38) having a hardware thread selector (see at least Fig.19) and an execution pipeline (see column 9, lines 36-38), the method comprising:

a) storing a first context of said processor in a first set of data storage devices comprising a first thread state corresponding to a first program thread. See column 20, lines 42-52, and note that periodic threads exist. Also, see column 9, lines 36-38, and note that the system is pipelined.

b) a first set of data storage devices capable of storing a first thread state of said processor. See Fig.10 and column 19, line 48, to column 20, line 36. Essentially a separate thread control block (TCB) exists for each thread to hold information pertaining to that thread. Information is also stored in CPU registers.

b) storing a second context of said processor in a second set of data storage devices comprising a second thread state corresponding to a second program thread. See Fig.10 and column 19, line 48, to column 20, line 36. Essentially a separate thread control block (TCB) exists for each thread to hold information pertaining to that thread. Information is also stored in CPU registers.

c) Gee has not explicitly taught switching the processor from the first thread state to the second thread state by coupling the execution pipeline from the first set of data storage devices to the second set of storage devices via the hardware thread selector at a fixed time according to a predetermined fixed execution schedule, said execution schedule specifying that the processor should switch to the first thread state every first number of cycles and that the processor should switch to the second thread state every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles. However, see Fig.19, column 20, lines 42-52, and column 32, lines 7-31, and note the "piano roll" scheduler for scheduling periodic interrupts. To summarize this scheduling concept, each column in Fig.19 corresponds to a thread assigned to a particular priority. For instance, column 3 represents a thread having a priority value of 3. And, column 29 represents a thread assigned a priority value of 29 (hereafter "thread 29"). Each row in Fig.19 represents a chord, where the N^{th} chord is "accessed" or "played" at the N^{th} interrupt triggered by the expiration of a hardware timer. So, for example, if a periodic interrupt timer is set to expire every 100 ns, then the first time it expires at 100 ns, row 0 will be accessed. At 200 ns, row 1 will be accessed. At 300 ns, row 2 will be accessed, and so on. In each row, bits may be set to enable periodic interrupts. For instance, in Fig.19, the user has set up thread 29 as being allocated processing time every time the timer expires (due to the entire thread 29 column being filled with ones). If the interrupt timer is set at 1 second, then thread 29

executes at a rate of 1 Hz (i.e., it is allocated processing time once a second, which corresponds to a number of cycles depending on the speed of the machine). Although a specific combination of periodic threads has not been taught, one of ordinary skill in the art would have recognized that setting up a fixed schedule of periodic threads in Gee is well within Gee's capability. Certainly, Gee could not reasonably be expected to teach every possible combination of periodic threads, but Gee's intended functionality is apparent to one of ordinary skill in the art. Assuming a timer interrupt of 1 Hz (for simplicity) and a cleared piano roll table, thread 21 could be set up such that it executes every other chord (so instead of its bit being set in every row, it could be set in every other row (0, 2, 4, 6, etc.), thereby giving it an execution frequency of 0.5 Hz, or an allocation rate of once every 2 seconds). Similarly, thread 19 could be set up such that it executes every 4th chord (so its bit will be set to '1' in rows 1, 5, 9, etc., thereby giving it an execution frequency of 0.25 Hz, or an allocation rate of once every 4 seconds). Finally, thread 17 could be set up such that it executes every 4th other chord as well, but starting in a different cycle (so its bit will be set to '1' in rows 3, 7, 11, etc., thereby giving it an execution frequency of 0.25 Hz, or an allocation rate of once every 4 seconds). With the piano roll table set in this manner, the execution of threads would be 21-19-21-17-21-19-21-17, etc. From this, it is clear that thread 21 is allocated processing time every X/4 cycles, where X is the number of cycles in 1 second. And, thread 19 is allocated processing time every X/2 cycles. Again, although this specific example was not taught by Gee, the functionality of Gee's piano table allows for such a configuration. To Gee, the specific examples were apparently not as important as setting forth the general concept behind the table. However, one would see the advantage of employing periodic threads to accomplish time-related tasks. For instance, in a real-time system, one might

want to poll for a particular button input every 5 seconds and read a temperature sensor every 10 seconds. Gee's system allows for such flexibility. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to configure Gee's piano table such that that the first thread is allocated processing time every first number of cycles and that the second thread is allocated processing time every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles.

17. Referring to claim 47, Gee, as modified, has taught a method as described in claim 46, wherein the switching comprises changing a state selection register included in the hardware thread selector. See Fig.10 and column 19, line 53, to column 20, line 36, and note that these registers are updated upon thread switches.

18. Claims 1, 29-33, 42-43, and 45-47 are rejected under 35 U.S.C. 103(a) as being unpatentable over Borkenhagen et al., U.S. Patent No. 6,076,157 (herein referred to as Borkenhagen) in view of Gee.

19. Referring to claim 1, Borkenhagen has taught a computer based system for switching between program contexts comprising:

a) a processor capable of having a first program thread and a second program thread in an execution pipeline having a thread selection hardware. See Fig.4A and note that the processor includes thread switch (selection) hardware which selects between a first thread (thread 0) and a second thread (thread 1). Also, see column 7, lines 15-20, and note that the processor has an execution pipeline.

b) a first set of data storage devices capable of storing a first thread state of said processor. See Fig.4A, component 442, and column 10, lines 18-56. Note that there is a first set of storage devices for storing a group of bits which represent the state of the first thread.

c) a second set of data storage devices capable of storing a second thread state of said processor. See Fig.4A, component 444, and column 10, lines 18-56. Note that there is a second set of storage devices for storing a group of bits which represent the state of the second thread.

d) Borkenhagen has not explicitly taught a hardware thread scheduler for identifying which of said program threads said processor executes and configurable to allocate available processing time of the processor among at least the first and second program threads by causing thread-switching at a fixed time according to a predetermined fixed schedule, said schedule specifying that the first thread should be allocated processing time every first number of cycles and that the second thread should be allocated processing time every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles. However, Gee has taught such a concept. See Fig.19, column 20, lines 42-52, and column 32, lines 7-31, and note the "piano roll" scheduler for scheduling periodic interrupts. To summarize this scheduling concept, each column in Fig.19 corresponds to a thread assigned to a particular priority. For instance, column 3 represents a thread having a priority value of 3. And, column 29 represents a thread assigned a priority value of 29 (hereafter "thread 29"). Each row in Fig.19 represents a chord, where the N^{th} chord is "accessed" or "played" at the N^{th} interrupt triggered by the expiration of a hardware timer. So, for example, if a periodic interrupt timer is set to expire every 100 ns, then the first time it expires at 100 ns, row 0 will be accessed. At 200 ns, row 1 will be accessed. At 300 ns, row 2 will be accessed, and so on. In each row, bits may be set to enable periodic

interrupts. For instance, in Fig.19, the user has set up thread 29 as being allocated processing time every time the timer expires (due to the entire thread 29 column being filled with ones). If the interrupt timer is set at 1 second, then thread 29 executes at a rate of 1 Hz (i.e., it is allocated processing time once a second, which corresponds to a number of cycles depending on the speed of the machine). Although a specific combination of periodic threads has not been taught, one of ordinary skill in the art would have recognized that setting up a fixed schedule of periodic threads in Gee is well within Gee's capability. Certainly, Gee could not reasonably be expected to teach every possible combination of periodic threads, but Gee's intended functionality is apparent to one of ordinary skill in the art. Assuming a timer interrupt of 1 Hz (for simplicity) and a cleared piano roll table, thread 21 could be set up such that it executes every other chord (so instead of its bit being set in every row, it could be set in every other row (0, 2, 4, 6, etc.), thereby giving it an execution frequency of 0.5 Hz, or an allocation rate of once every 2 seconds). Similarly, thread 19 could be set up such that it executes every 4th chord (so its bit will be set to '1' in rows 1, 5, 9, etc., thereby giving it an execution frequency of 0.25 Hz, or an allocation rate of once every 4 seconds). Finally, thread 17 could be set up such that it executes every 4th other chord as well, but starting in a different cycle (so its bit will be set to '1' in rows 3, 7, 11, etc., thereby giving it an execution frequency of 0.25 Hz, or an allocation rate of once every 4 seconds). With the piano roll table set in this manner, the execution of threads would be 21-19-21-17-21-19-21-17, etc. From this, it is clear that thread 21 is allocated processing time every X/4 cycles, where X is the number of cycles in 1 second. And, thread 19 is allocated processing time every X/2 cycles. Again, although this specific example was not taught by Gee, the functionality of Gee's piano table allows for such a configuration. To Gee, the specific

examples were apparently not as important as setting forth the general concept behind the table. However, one would see the advantage of employing periodic threads to accomplish time-related tasks. For instance, in a real-time system, one might want to poll for a particular button input every 5 seconds and read a temperature sensor every 10 seconds. Gee's system allows for such flexibility. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to configure Gee's piano table such that that the first thread is allocated processing time every first number of cycles and that the second thread is allocated processing time every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles. And, it would have been further obvious to modify Borkenhagen to include the piano scheduling of Gee so that thread may be scheduled periodically. One would have been motivated to make such a modification to Borkenhagen in order to, at the very least, increase the scheduling flexibility of Borkenhagen.

20. Referring to claim 29, Borkenhagen in view of Gee has taught a system as described in claim 1. Borkenhagen has further taught that said thread selection hardware in the pipelined processor switches between said first and second thread state after the end of the execution of a first program instruction in the first thread and before the beginning of the execution of a second program instruction. This is deemed inherent because thread A will execute for some amount of time and then a switch will occur to another thread. The switching marks the end of executing an instruction from thread A and the beginning of executing an instruction in thread B. And, clearly, the system must be in the second state before it can begin executing the second thread.

21. Referring to claim 30, Borkenhagen in view of Gee has taught a system as described in claim 1. Borkenhagen has further taught that said processor is an embedded pipelined processor. See claim 14 of Borkenhagen. The processor, which is pipelined, is embedded in a system.

22. Referring to claim 31, Borkenhagen in view of Gee has taught a system as described in claim 1. Borkenhagen has further taught that said first state is the state of the processor during the execution of the first program thread. See column 10, lines 18-56.

23. Referring to claim 32, Borkenhagen in view of Gee has taught a system as described in claim 1. Borkenhagen has further taught that said second state is the state of the processor during the execution of the second program thread. See column 10, lines 18-56.

24. Referring to claim 33, Borkenhagen in view of Gee has taught a system as described in claim 1. Gee has further taught that said processor switches between said first and second state by changing a state selection register. See column 20, lines 42-52, and column 32, lines 7-31. The hardware timer register (state selection register) is decremented each cycle until it gets to zero and then a thread switch occurs. Therefore, the processor switches between first and second state by changing a state selection register.

25. Referring to claim 42, Borkenhagen in view of Gee has taught a system as described in claim 1. Borkenhagen has further taught that said processor is capable of restoring said second state of said processor during execution of said first program thread. See Fig.4A, component 450, and note that at some point during execution of the first thread, a thread switch will occur, and the second state is restored. That is, the switching to (restoring the state of) the second thread occurs during execution of the first thread.

26. Referring to claim 43, Borkenhagen in view of Gee has taught a system as described in claim 1. Borkenhagen has further taught that said processor is capable of storing said second thread state of said processor during execution of said first program thread. See column 13, lines 20-45. This control register (and control state) allows the system to specify which types of events would result in the switching of the associated thread, thereby increasing flexibility by allowing the user to choose how the thread may or may not be switched. This register may be set by any thread for itself or another thread.

27. Referring to claim 45, Borkenhagen in view of Gee has taught a system as described in claim 1. Gee has further taught that the predetermined fixed schedule is one of a fixed strict schedule, a semi-flexible strict schedule, and a loose strict schedule. Gee has taught at least a fixed strict schedule where threads are switched based on the data in Fig.19.

28. Referring to claim 46, Borkenhagen has taught a computer based method for switching between program contexts in a multithreading pipelined processor (see Fig.4A and the abstract) having a hardware thread selector (see Fig.4A) and an execution pipeline (see column 7, lines 15-20 and note that the processor has an execution pipeline), the method comprising:

a) storing a first context of said processor in a first set of data storage devices comprising a first thread state corresponding to a first program thread. See Fig.4A, component 442, and column 10, lines 18-56. Note that there is a first set of storage devices for storing a group of bits which represent the state of the first thread.

b) storing a second context of said processor in a second set of data storage devices comprising a second thread state corresponding to a second program thread. See Fig.4A, component 444, and

column 10, lines 18-56. Note that there is a second set of storage devices for storing a group of bits which represent the state of the second thread.

c) Borkenhagen has not explicitly taught switching the processor from the first thread state to the second thread state by coupling the execution pipeline from the first set of data storage devices to the second set of storage devices via the hardware thread selector at a fixed time according to a predetermined fixed execution schedule, said execution schedule specifying that the processor should switch to the first thread state every first number of cycles and that the processor should switch to the second thread state every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles. However, Gee has taught such a concept. See Fig.19, column 20, lines 42-52, and column 32, lines 7-31, and note the "piano roll" scheduler for scheduling periodic interrupts. To summarize this scheduling concept, each column in Fig.19 corresponds to a thread assigned to a particular priority. For instance, column 3 represents a thread having a priority value of 3. And, column 29 represents a thread assigned a priority value of 29 (hereafter "thread 29"). Each row in Fig.19 represents a chord, where the N^{th} chord is "accessed" or "played" at the N^{th} interrupt triggered by the expiration of a hardware timer. So, for example, if a periodic interrupt timer is set to expire every 100 ns, then the first time it expires at 100 ns, row 0 will be accessed. At 200 ns, row 1 will be accessed. At 300 ns, row 2 will be accessed, and so on. In each row, bits may be set to enable periodic interrupts. For instance, in Fig.19, the user has set up thread 29 as being allocated processing time every time the timer expires (due to the entire thread 29 column being filled with ones). If the interrupt timer is set at 1 second, then thread 29 executes at a rate of 1 Hz (i.e., it is allocated processing time once a second, which corresponds to a number of cycles depending on the speed of the

machine). Although a specific combination of periodic threads has not been taught, one of ordinary skill in the art would have recognized that setting up a fixed schedule of periodic threads in Gee is well within Gee's capability. Certainly, Gee could not reasonably be expected to teach every possible combination of periodic threads, but Gee's intended functionality is apparent to one of ordinary skill in the art. Assuming a timer interrupt of 1 Hz (for simplicity) and a cleared piano roll table, thread 21 could be set up such that it executes every other chord (so instead of its bit being set in every row, it could be set in every other row (0, 2, 4, 6, etc.), thereby giving it an execution frequency of 0.5 Hz, or an allocation rate of once every 2 seconds). Similarly, thread 19 could be set up such that it executes every 4th chord (so its bit will be set to '1' in rows 1, 5, 9, etc., thereby giving it an execution frequency of 0.25 Hz, or an allocation rate of once every 4 seconds). Finally, thread 17 could be set up such that it executes every 4th other chord as well, but starting in a different cycle (so its bit will be set to '1' in rows 3, 7, 11, etc., thereby giving it an execution frequency of 0.25 Hz, or an allocation rate of once every 4 seconds). With the piano roll table set in this manner, the execution of threads would be 21-19-21-17-21-19-21-17, etc. From this, it is clear that thread 21 is allocated processing time every X/4 cycles, where X is the number of cycles in 1 second. And, thread 19 is allocated processing time every X/2 cycles. Again, although this specific example was not taught by Gee, the functionality of Gee's piano table allows for such a configuration. To Gee, the specific examples were apparently not as important as setting forth the general concept behind the table. However, one would see the advantage of employing periodic threads to accomplish time-related tasks. For instance, in a real-time system, one might want to poll for a particular button input every 5 seconds and read a temperature sensor every 10 seconds. Gee's system allows for such

flexibility. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to configure Gee's piano table such that that the first thread is allocated processing time every first number of cycles and that the second thread is allocated processing time every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles. And, it would have been further obvious to modify Borkenhagen to include the piano scheduling of Gee so that thread may be scheduled periodically. One would have been motivated to make such a modification to Borkenhagen in order to, at the very least, increase the scheduling flexibility of Borkenhagen.

29. Referring to claim 47, Borkenhagen in view of Gee has taught a method as described in claim 46. Gee has further taught that the switching comprises changing a state selection register included in the hardware thread selector. See column 20, lines 42-52, and column 32, lines 7-31. The hardware timer register (state selection register) is decremented each cycle until it gets to zero and then a thread switch occurs. Therefore, the processor switches between first and second state by changing a state selection register.

30. Claims 2-4, 13, 16-17, 19-24, and 56-57 are rejected under 35 U.S.C. 103(a) as being unpatentable over Joy et al., U.S. Patent No. 6,542,991 (herein referred to as Joy) in view of McCrackin et al., "Using Horizontal Prefetching to Circumvent the Jump Problem", 1991 (herein referred to as McCrackin). In addition, the Free Online Dictionary of Computing (herein referred to as FOLDOC) is cited as extrinsic evidence for showing that "cycle" refers to a clock cycle.

31. Referring to claim 17, Joy has taught a computer based system for switching between program contexts comprising:

a) a pipelined processor (Fig.3, component 300; column 8, lines 14-67) capable of having a first program thread and a second program thread in an execution pipeline having a thread selection hardware (see Fig.6 and note the "thread select logic"; column 13 lines 5-23, column 15 lines 4-7), the execution pipeline including a set of stages for executing instructions and configured to execute a single instruction at each different stage in the set of stages (see claim 1, for instance, and note the existence of a pipeline). It should be noted that pipelines inherently include stages which perform different tasks, and in which a different instruction may be executing at one time.

b) a first set of data storage devices capable of storing a first thread state of said pipelined processor. See Fig.3, components 310 and 330, and column 8, lines 27-44. Note that component 310 includes flip-flops and a register file structure for storing a first thread state (for thread 0). Also, instruction cache 330 stores instructions for thread 0.

c) a second set of data storage devices capable of storing a second thread state of said pipelined processor. See Fig.3, components 312 and 330, and column 8, lines 27-44. Note that component 312 includes flip-flops and a register file structure for storing a second thread state (for thread 1). Also, instruction cache 330 stores instructions for thread 1. Note that thread 0 and thread instructions are each stored in a unique location in memory (clearly two sets of data cannot be stored in the same physical location).

d) a hardware thread scheduler for identifying which of said program threads said pipelined processor executes (Fig.6; column 15, lines 4-7) and configurable to allocate available processing time of the pipelined processor among at least the first and second program threads

according to an execution schedule by controlling whether the execution pipeline retrieves instructions from the first set of data storage devices or the second set of data storage devices. See column 3, lines 33-51, and note that Joy's thread-switching may be of the oblivious type, in which threads are switched every N cycles without notification of stalling. Note that depending on the thread schedule, the appropriate data storage device is selected to retrieve instructions from.

e) wherein said thread selection hardware in the pipelined processor switches from said first thread state to said second thread state between consecutive instruction cycles in response to the hardware thread scheduler identifying which of said program threads said pipelined processor executes. Recall from above that Joy has taught thread switching. An example is shown in column 3, lines 33-51, and column 17, lines 1-6). It is deemed inherent that switching occurs between consecutive instruction cycles because a switch will always occur between a last instruction of a first thread and a first instruction of a second thread. Since applicant has limited "instruction cycle" as the time involved in processing an actual instruction (fetching, decoding, execution, etc.), the switch can only happen between two consecutive instruction cycles. That is, a first instruction cycle is associated with the last instruction of the first thread and a second instruction cycle is associated with a first instruction of the second thread. The switch will occur between these two consecutive instruction cycles, and the switch itself does not occur in an instruction cycle because a switch is not an instruction that is fetched, decoded, executed, etc. It is merely caused by an internal signal which results from a hardware counter reaching a desired count, for instance. To further illustrate, assume Ia is an instruction cycle associated with an instruction in the first thread (the first thread being switched out) and Ib is an instruction cycle

associated with an instruction in the second thread (the second thread being switched in). According to Joy, column 16, lines 61-66, a thread switch takes 3 clock cycles or less (note they're not instruction cycles, which has a specific meaning in the art. "Cycle" used by itself implies a clock cycle as taught by FOLDOC (see the attached definition of "cycle"). Certainly, if Joy meant for switching to take 3 instruction cycles or less, given that "instruction cycle" has a particular meaning, then Joy would have disclosed "instruction cycles" and not just "cycles"). If a clock cycle of delay is represented by a '-', then a simplified thread switch timeline appears as follows:

Ia---Ib

Note that there are only two instruction cycles shown (Ia and Ib). They are consecutive, and the switching occurs in between. Again, note that the switching is not part of an instruction cycle since the switching is not part of an instruction.

f) Joy has not taught that the processor switches from said first thread state to said second thread state between consecutive instruction cycles without incurring a time penalty. However, McCrackin has taught such a concept. See page 1287, column 2, section II, paragraphs 2-3. Specifically, resources of the execution and fetch units are duplicated for as many thread contexts that exist. Such duplication allows for elimination of context switching overhead because instead of having to spend time saving and restoring state information in shared resources each time a context is switched (as is the case in Joy, column 15, lines 1-7), one of the duplicated (non-shared) contexts is merely selected to switch threads, i.e., no saving/restoring in shared resources is required. As a result, in order to eliminate time penalties associated with thread switching, it would have been obvious to one of ordinary skill in the art at the time of the

invention to modify Joy such that resources are duplicated for each thread instead of sharing resources and having to perform time-costly saves and restores each time a thread switch occurs.

32. Referring to claim 2, Joy in view of McCrackin has taught a system as described in claim 17, wherein said first thread state is the thread state of the processor during the execution of the first program thread. See Joy, Fig.3, components 310 and 330, and column 8, lines 27-44. Note that component 310 includes flip-flops and a register file structure for storing a first thread state (for thread 0).

33. Referring to claims 3, Joy in view of McCrackin has taught a system as described in claim 17, wherein said second thread state is the thread state of the processor during the execution of the second program thread. See Joy, Fig.3, components 312 and 330, and column 8, lines 27-44. Note that component 312 includes flip-flops and a register file structure for storing a second thread state (for thread 1).

34. Referring to claim 4, Joy in view of McCrackin has taught a system as described in claim 17, wherein said processor switches between said first and second thread state by changing a state selection register. See Joy, Fig.5 and column 13, lines 5-64. The thread select logic includes a flip-flop for each thread, where the flip-flops collectively form a register that includes an active bit in the position corresponding to the active thread.

35. Referring to claim 13, Joy in view of McCrackin has taught a system as described in claim 17, wherein said processor is capable of restoring said second thread state of said processor during execution of said first program thread. See Joy, column 6, lines 15-35. Clearly, a thread is executed until a switch signal is given. Therefore, switching to (restoring a) thread occurs during execution of another thread.

36. Referring to claim 16, Joy has taught a system as described in claim 17, wherein the execution schedule is one of a fixed strict schedule, a semi-flexible strict schedule, and a loose strict schedule. From column 3, lines 28-51, switching every N cycles is considered a fixed strict schedule, i.e., a thread switch must occur every N cycles.

37. Referring to claim 19, Joy has taught a computer based method for switching between program contexts in a multithreading pipelined processor (Fig.3; column 8, lines 14-67) having a hardware thread selector (Fig.6) and an execution pipeline (see claim 1 of Joy, for instance), the execution pipeline including a set of stages for executing instructions and configured to execute a single instruction at each different stage of the set of stages (it should be noted that pipelines inherently include stages which perform different tasks, and in which a different instruction may be executing at one time), the method comprising:

a) storing a first context of said pipelined processor in a first set of data storage devices, the first context corresponding to a first program thread. See Fig.3, components 310 and 330, and column 8, lines 27-44. Note that component 310 includes flip-flops and a register file structure for storing a first thread state (for thread 0).

b) storing a second context of said pipelined processor in a second set of data storage devices, the second context corresponding to a second program thread. See Fig.3, components 312 and 330, and column 8, lines 27-44. Note that component 312 includes flip-flops and a register file structure for storing a second thread state (for thread 1).

c) switching the pipelined processor from executing the first program thread to executing the second program thread between the end of an execution cycle and before the beginning of a next consecutive execution cycle by coupling the execution pipeline from the first set of data storage

devices to the second set of storage devices via the hardware thread selector. Recall from above that Joy has taught thread switching. An example is shown in column 3, lines 33-51, and column 17, lines 1-6). It is deemed inherent that switching occurs between consecutive instruction cycles because a switch will always occur between a last instruction of a first thread and a first instruction of a second thread. Since applicant has limited "instruction cycle" as the time involved in processing an actual instruction (fetching, decoding, execution, etc.), the switch can only happen between two consecutive instruction cycles. That is, a first instruction cycle is associated with the last instruction of the first thread and a second instruction cycle is associated with a first instruction of the second thread. The switch will occur between these two consecutive instruction cycles, and the switch itself does not occur in an instruction cycle because a switch is not an instruction that is fetched, decoded, executed, etc. It is merely caused by an internal signal which results from a hardware counter reaching a desired count, for instance. To further illustrate, assume Ia is an instruction cycle associated with an instruction in the first thread (the first thread being switched out) and Ib is an instruction cycle associated with an instruction in the second thread (the second thread being switched in). According to Joy, column 16, lines 61-66, a thread switch takes 3 clock cycles or less (note they're not instruction cycles, which has a specific meaning in the art. "Cycle" used by itself implies a clock cycle as taught by FOLDOC (see the attached definition of "cycle"). Certainly, if Joy meant for switching to take 3 instruction cycles or less, given that "instruction cycle" has a particular meaning, then Joy would have disclosed "instruction cycles" and not just "cycles"). If a clock cycle of delay is represented by a '-', then a simplified thread switch timeline appears as follows:

Ia---Ib

Note that there are only two instruction cycles shown (Ia and Ib). They are consecutive, and the switching occurs in between. Again, note that the switching is not part of an instruction cycle since the switching is not part of an instruction.

d) Joy has not taught switching the pipelined processor from executing the first program thread to executing the second program thread between the end of an execution cycle and before the beginning of a next consecutive execution cycle without incurring a time penalty. However, McCrackin has taught such a concept. See page 1287, column 2, section II, paragraphs 2-3. Specifically, resources of the execution and fetch units are duplicated for as many thread contexts that exist. Such duplication allows for elimination of context switching overhead because instead of having to spend time saving and restoring state information in shared resources each time a context is switched (as is the case in Joy, column 15, lines 1-7), one of the duplicated (non-shared) contexts is merely selected to switch threads, i.e., no saving/restoring in shared resources is required. As a result, in order to eliminate time penalties associated with thread switching, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Joy such that resources are duplicated for each thread instead of sharing resources and having to perform time-costly saves and restores each time a thread switch occurs.

38. Referring to claim 20, Joy in view of McCrackin has taught a system as described in claim 19, wherein the switching comprises changing a state selection register included in the hardware thread selector. See Joy, Fig.5 and column 13, lines 5-64. The thread selector logic includes a flip-flop for each thread, where the flip-flops collectively form a register that includes an active bit in the position corresponding to the active thread. When a thread is switched, then a new bit in the register would be set while the previously set bit is reset.

39. Referring to claim 21, Joy in view of McCrackin has taught a method as described in claim 19, further comprising identifying which of the said program threads said processor executes according to an execution schedule. See Joy, column 3, lines 33-51.

40. Referring to claim 22, Joy in view of McCrackin has taught a method as described in claim 21, further comprising allocating available processing time of the processor among at least the first and second threads according to the execution schedule. See Joy, column 3, lines 33-51.

41. Referring to claim 23, Joy in view of McCrackin has taught a method as described in claim 22, wherein the allocating comprises dividing the available execution time into a plurality of quanta, each quanta corresponding to a number of instruction cycles for execution of a thread. See Joy, column 3, lines 33-51, and note that each thread will execute for N cycles in one scheduling embodiment.

42. Referring to claim 24, Joy in view of McCrackin has taught a method as described in claim 23, wherein at least one quanta corresponds to a thread that is scheduled to execute periodically after a fixed number of execution cycles. See Joy, column 3, lines 33-36.

43. Referring to claim 56, Joy in view of McCrackin has taught the method of claim 19, wherein switching the pipelined processor from executing the first program thread to executing the second program thread between the end of an execution cycle and before the beginning of a next consecutive execution cycle without incurring a time penalty comprises: identifying a context number associated with the second program thread, the context number identifying the second set of data storage devices; communicating the context number associated with the second program thread to the execution pipeline; and loading instructions from the second set of data storage devices into the execution pipeline. The examiner asserts that each of these steps

are inherent in the combined Joy and McCrackin system. If the system switches contexts, then a signal selecting a second context is inherently issued such that the second thread's instructions may be retrieved and loaded into the pipeline. If N contexts exist, as is the case in McCrackin, then one of N threads must be specified.

44. Referring to claim 57, Joy in view of McCrackin has taught the system of claim 17, wherein said thread selection hardware in the pipelined processor switches from said first thread state to said second thread state between consecutive instruction cycles without incurring a time penalty comprises: identifying a context number associated with the second program thread, the context number identifying the second set of data storage devices; communicating the context number associated with the second program thread to the execution pipeline; and loading instructions from the second set of data storage devices into the execution pipeline. The examiner asserts that each of these steps are inherent in the combined Joy and McCrackin system. If the system switches contexts, then a signal selecting a second context is inherently issued such that the second thread's instructions may be retrieved and loaded into the pipeline. If N contexts exist, as is the case in McCrackin, then one of N threads must be specified.

45. Claims 5-12, 18, and 25-28 are rejected under 35 U.S.C. 103(a) as being unpatentable over Joy in view of McCrackin and further in view of Ramakrishnan et al., U.S Patent No. 6,085,215 (herein referred to as Ramakrishnan).

46. Referring to claim 5, Joy in view of McCrackin has taught a system as described in claim 17. Joy has not taught that said hardware thread scheduler includes a thread identifier for identifying at least one hard-real-time (HRT) thread and at least one non-real-time thread and a

HRT scheduler for regularly scheduling said HRT thread in available time quanta such that said HRT thread is scheduled to ensure the execution of the HRT in a predetermined time. However, Ramakrishnan has taught such a concept. See column 4, line 52, to column 5, line 3, and the abstract. Note that real-time and general (non real-time) threads are determined and that a real time thread is scheduled during the available time quanta such that it executes in predetermined time. In such a system, real-time threads, which perform time-critical tasks, are given priority over general threads. This is clear because the general threads execute for a minimum time and then after that, if a real-time thread needs processing, then it will preempt that general thread. Furthermore, Ramakrishnan has taught that such a system prevents starvation of threads (since all threads get some time to process) and offers a greater degree of fairness in allocating processing resources to various tasks. See column 4, lines 11-14. As a result, in order to execute time-critical tasks in a non-starving and fair way, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Joy to be used in a time-critical environment and to include at least one HRT thread and an HRT scheduler, as taught by Ramakrishnan.

47. Referring to claim 6, Joy in view of McCrackin in view of Ramakrishnan has taught a system as described in claim 5. Ramakrishnan has further taught that said time quanta is at least one instruction cycle. See the abstract and note that real-time threads are scheduled for a preselected maximum amount of time. This time is inherently at least one cycle because if it were any less (zero cycles), then the thread would never execute.

48. Referring to claim 7, Joy in view of McCrackin in view of Ramakrishnan has taught a system as described in claim 5. Ramakrishnan has further taught that said hardware thread scheduler schedules a non-real-time (NRT) thread to replace a scheduled HRT thread if said

HRT is idle. See Fig.3, step 74, and note that if a real-time thread is idle (has no work), then the schedule is free to replace it with another thread, where another thread is either an NRT or another HRT.

49. Referring to claim 8, Joy in view of McCrackin in view of Ramakrishnan has taught a system as described in claim 5. Ramakrishnan has further taught that said thread scheduler schedules the execution of non-real-time (NRT) threads in quanta not allocated to HRT threads. See the Fig.2A and note that a real-time thread is allocated time and that time is the real-time thread's time (step 52). After the HRT is done, an NRT may be scheduled. That is, they are both not scheduled in the same quanta (at the same time). The HRT is to execute (assuming the HRT is first to execute), and then the NRT is to execute.

50. Referring to claim 9, Joy in view of McCrackin in view of Ramakrishnan has taught a system as described in claim 8. Ramakrishnan has further taught that said thread scheduler regularly schedules NRT threads to be executed. See the abstract and note that there may be a plurality of NRTs for scheduling. They are scheduled for minimum times throughout the entire execution process.

51. Referring to claim 10, Joy in view of McCrackin in view of Ramakrishnan has taught a system as described in claim 5. Joy has further taught:

a) a first storage device for storing program instructions, said processor fetching instructions from the first storage device within a first fetch period. See Fig.3, component 330. The instruction cache (I$) will be fetched from during the time that the instructions needed are in the cache.

b) a second storage device for storing program instructions, said processor fetching instructions from the second storage device within a second fetch period. See column 9, line 66. The main memory will be fetched from during the time that the instructions needed are not in the cache. c) wherein said first fetch period is substantially shorter than said second fetch period. Fetching from a cache is shorter than fetching from main memory, as is known in the art.

52. Referring to claim 11, Joy in view of McCrackin in view of Ramakrishnan has taught a system as described in claim 10. Joy has not taught that said first storage device for storing program instructions comprises a static RAM. However, Official Notice is taken that virtually all caches are implemented with static RAM (SRAM) and that SRAM and its advantages are well known and accepted in the art. SRAM is fast, which makes it suitable for caches, and unlike DRAM, it does not need to be refreshed in order to maintain its contents. Consequently, for speed and storage ability, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Joy's instruction cache such that it is implemented in SRAM.

53. Referring to claim 12, Joy in view of McCrackin in view of Ramakrishnan has taught a system as described in claim 10. Joy has not taught that said second storage device for storing program instructions comprises a flash memory. However, Official Notice is taken that flash-based main memories and their advantages are well known and accepted in the art. A computer hierarchy based upon volatile main memory loses all information in main memory when power is turned off. A flash-based non-volatile main memory, however, reduces or eliminates the lengthy process of obtaining information from disk when power is turned on. Therefore flash main memory based computer system has higher system performance when a program is initially executed than would a volatile main memory based computer system. As a result, it would have

been obvious to one of ordinary skill in the art at the time of the invention to modify Joy's main memory such that it is a flash-based main memory.

54. Referring to claim 18, Joy in view of McCrackin in view of Ramakrishnan has taught a system as described in claim 5. Joy has further taught that said time quanta is exactly one instruction cycle. See column 17, lines 1-6, and note that Joy has taught a time quanta of N cycles, where N includes 1.

55. Referring to claim 25, Joy in view of McCrackin has taught a method as described in claim 21. Joy has not taught identifying at least one hard real-time (HRT) thread and at least one non real-time (NRT) thread. However, Ramakrishnan has taught the concept of real-time threads and general (non-real-time). See column 4, line 52, to column 5, line 3, and the abstract. Note that real-time threads are identified and are those that perform time-critical work and therefore should be given priority in the system over general threads. This priority concept is clear in Ramakrishnan because the general threads execute for a minimum time and then after that, if a real time thread needs processing, then it will preempt that general thread. Furthermore, Ramakrishnan has taught that such a system gets important work done while preventing starvation of threads (since all threads get some time to process) and offering a greater degree of fairness in allocating processing resources to various tasks. See column 4, lines 11-14. As a result, in order to execute time-critical tasks in a non-starving and fair way, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Joy to identify at least one HRT and at least one NRT thread, as taught by Ramakrishnan.

56. Referring to claim 26, Joy in view of McCrackin in view of Ramakrishnan has taught a method as described in claim 25. Ramakrishnan has further taught scheduling the HRT thread in

available time quanta such that said HRT thread is scheduled to ensure the execution of the HRT thread within a predetermined time. See column 4, line 52, to column 5, line 3, and the abstract. Note that HRT threads are given a maximum time in which to execute. This ensures the execution of the HRT within that maximum time.

57. Referring to claim 27, Joy in view of McCrackin in view of Ramakrishnan has taught a method as described in claim 25. Ramakrishnan has further taught scheduling an NRT thread for a quantum allocated for an HRT thread if said HRT thread is idle. See Fig.3, step 74, and note that if a real-time thread is idle (has no work), then the schedule is free to replace it with another thread, where another thread is either an NRT or another HRT.

58. Referring to claim 28, Joy in view of McCrackin in view of Ramakrishnan has taught a method as described in claim 25. Ramakrishnan has further taught scheduling NRT threads in quanta not allocated for HRT threads. See the Fig.2A and note that a real-time thread is allocated time and that time is the real-time thread's time (step 52). After the HRT is done, an NRT may be scheduled. That is, they are both not scheduled in the same quanta (at the same time). The HRT is to execute (assuming the HRT is first to execute), and then the NRT is to execute.

59. Claims 34-41 and 48-55 are rejected under 35 U.S.C. 103(a) as being unpatentable over Gee in view of Ramakrishnan.

60. Referring to claim 34, Gee, as modified, has taught a system as described in claim 1. Gee has not taught that said hardware thread scheduler includes a thread identifier for identifying at least one hard-real-time (HRT) thread and at least one non-real-time thread and a HRT scheduler

for regularly scheduling said HRT thread in available time quanta such that said HRT thread is scheduled to ensure the execution of the HRT in a predetermined time. However, Ramakrishnan has taught such a concept. See column 4, line 52, to column 5, line 3, and the abstract. Note that real-time and general (non real-time) threads are determined and that a real time thread is scheduled during the available time quanta such that it executes in predetermined time (i.e., in a preselected maximum time). In such a system, real-time threads, which perform time-critical tasks, are given priority over general threads. This is clear because the general threads execute for a minimum time and then after that, if a real time thread needs processing, then it will preempt that general thread. Furthermore, Ramakrishnan has taught that such a system prevents starvation of threads (since all threads get some time to process) and offers a greater degree of fairness in allocating processing resources to various tasks. See column 4, lines 11-14. As a result, in order to execute time-critical tasks in a non-starving and fair way, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Gee to be used in a time-critical environment and to include at least one HRT thread and an HRT scheduler, as taught by Ramakrishnan. And, one would be motivated to use Gee in such an environment because Gee discusses real-time systems. See the title and background of invention.

61. Referring to claim 35, Gee in view of Ramakrishnan has taught a system as described in claim 34, wherein said time quanta is at least one instruction cycle. See the abstract of Ramakrishnan and note that real-time threads are scheduled for a preselected maximum amount of time. This time is inherently at least one cycle because if it were any less (zero cycles), then the thread would never execute.

62. Referring to claim 36, Gee in view of Ramakrishnan has taught a system as described in claim 34, wherein said hardware thread scheduler schedules a non-real-time (NRT) thread to replace a scheduled HRT thread if said HRT is idle. See Ramakrishnan, Fig.3, step 74, and note that if a real-time thread is idle (has no work), then the schedule is free to replace it with another thread, where another thread is either an NRT or another HRT.

63. Referring to claim 37, Gee in view of Ramakrishnan has taught a system as described in claim 34, wherein said hardware thread scheduler schedules the execution of non-real-time (NRT) threads in quanta not allocated to HRT threads. See Fig.2A of Ramakrishnan and note that a real-time thread is allocated time and that time is the real-time thread's time (step 52). After the HRT is done, an NRT may be scheduled. That is, they are both not scheduled in the same quanta (at the same time). The HRT is to execute (assuming the HRT is first to execute), and then the NRT is to execute.

64. Referring to claim 38, Gee in view of Ramakrishnan has taught a system as described in claim 37, wherein said hardware thread scheduler regularly schedules NRT threads to be executed. See the abstract of Ramakrishnan and note that there may be a plurality of NRTs for scheduling.

65. Referring to claim 39, Gee in view of Ramakrishnan has taught a system as described in claim 34.

a) Gee has not taught a first storage device for storing program instructions, said processor fetching instructions from the first storage device within a first fetch period. However, the examiner asserts that instruction caches are well known and advantageous in the art. Caches are high-speed memories that store the most recently accessed instructions. Therefore, in order to

quickly fetch instructions that have recently been accessed, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Gee to include an instruction cache as the first storage device.

b) Gee has taught a second storage device for storing program instructions, said processor fetching instructions from the second storage device within a second fetch period. See Fig.1, component 104. Main memory is known to hold instructions and is fetched from during the time that the instructions needed are not in the cache.

c) Gee, as modified, has taught that said first fetch period is substantially shorter than said second fetch period. Fetching from a cache takes less time than fetching from main memory. This is the very reason for employing a cache.

66. Referring to claim 40, Gee in view of Ramakrishnan has taught a system as described in claim 39. Gee has not taught that said first storage device for storing program instructions comprises a static RAM. However, Official Notice is taken that virtually all caches are implemented with static RAM (SRAM) and that SRAM and its advantages are well known and accepted in the art. SRAM is fast, which makes it suitable for caches, and unlike DRAM, it does not need to be refreshed in order to maintain its contents. Consequently, for speed and storage ability, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Gee's instruction cache such that it is implemented in SRAM.

67. Referring to claim 41, Gee in view of Ramakrishnan has taught a system as described in claim 39. Gee has not taught that said second storage device for storing program instructions comprises a flash memory. However, Official Notice is taken that flash-based main memories and their advantages are well known and accepted in the art. A computer hierarchy based upon

volatile main memory loses all information in main memory when power is turned off. A flash-based non-volatile main memory, however, reduces or eliminates the lengthy process of obtaining information from disk when power is turned on. Therefore flash main memory based computer system has higher system performance when a program is initially executed than would a volatile main memory based computer system. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Gee's main memory such that it is a flash-based main memory.

68. Referring to claim 48, Gee, as modified, has taught a method as described in claim 46. Gee has not taught identifying which of the said program threads said processor executes according to a hard real-time (HRT) execution schedule. However, Ramakrishnan has taught the concept of real-time threads. See column 4, line 52, to column 5, line 3, and the abstract. Note that real-time threads are identified and are those that perform time-critical work and therefore should be given priority in the system. In such a system, real-time threads are given priority over general threads (note that Gee has also taught using priority with threads in Fig.19). This priority concept is clear in Ramakrishnan because the general threads execute for a minimum time and then after that, if a real time thread needs processing, then it will preempt that general thread. Furthermore, Ramakrishnan has taught that such a system prevents starvation of threads (since all threads get some time to process) and offers a greater degree of fairness in allocating processing resources to various tasks, while allowing important work to get done. See column 4, lines 11-14. As a result, in order to execute time-critical tasks in a non-starving and fair way, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify

Gee to be used in a time-critical environment and to identify which of the program threads the processor executes according to an HRT execution schedule, as taught by Ramakrishnan.

69. Referring to claim 49, Gee in view of Ramakrishnan has taught a method as described in claim 48, further comprising allocating available processing time of the processor among at least the first and second threads according to the predetermined fixed execution schedule. Please recall the rejection of claim 46 and note that time is allocated to periodic threads according to the schedule fixed by the structure of Fig.19.

70. Referring to claim 50, Gee in view of Ramakrishnan has taught a method as described in claim 49, wherein the allocating comprises dividing the available execution time into a plurality of quanta, each quanta corresponding to a number of instruction cycles for execution of a thread. Again, see the rejection of claim 46. Each thread is allocated a number of cycles (quanta) in which to execute. This number is associated with the timeout value of the hardware timer.

71. Referring to claim 51, Gee in view of Ramakrishnan has taught a method as described in claim 50, wherein at least one quantum corresponds to a thread that is scheduled to execute periodically after a fixed number of execution cycles. Please recall the rejection of claim 46 and Fig.19 of Gee.

72. Referring to claim 52, Gee in view of Ramakrishnan has taught a method as described in claim 48, wherein identifying further comprises identifying at least one hard real-time (HRT) thread and at least one non real-time (NRT) thread. See Ramakrishnan, column 4, line 52, to column 5, line 3, and the abstract. Note that real-time threads are identified and are those that perform time-critical work and therefore should be given priority in the system over general threads, which are the less time-critical threads.

73. Referring to claim 53, Gee in view of Ramakrishnan has taught a method as described in claim 52, further comprising scheduling the HRT thread in available time quanta such that said HRT thread is scheduled to ensure the execution of the HRT thread within a predetermined time. See Ramakrishnan, column 4, line 52, to column 5, line 3, and the abstract. Note that HRT threads are given a maximum time in which to execute. This ensures the execution of the HRT within that maximum time.

74. Referring to claim 54, Gee in view of Ramakrishnan has taught a method as described in claim 52, further comprising scheduling an NRT thread for a quantum allocated for an HRT thread if said HRT thread is idle. See Ramakrishnan, Fig.3, step 74, and note that if a real-time thread is idle (has no work), then the schedule is free to replace it with another thread, where another thread is either an NRT or another HRT.

75. Referring to claim 55, Gee in view of Ramakrishnan has taught a method as described in claim 52, further comprising scheduling NRT threads in quanta not allocated for HRT threads. See Fig.2A of Ramakrishnan and note that a real-time thread is allocated time and that time is the real-time thread's time (step 52). After the HRT is done, an NRT may be scheduled. That is, they are both not scheduled in the same quanta (at the same time). The HRT is to execute (assuming the HRT is first to execute), and then the NRT is to execute.

76. Claim 14 is rejected under 35 U.S.C. 103(a) as being unpatentable over Joy in view of McCrackin and further in view of Borkenhagen.

77. Referring to claim 14, Joy in view of McCrackin has taught a system as described in claim 17. Joy has not explicitly taught that said processor is capable of storing said second

thread state of said processor during execution of said first program thread. However, Borkenhagen has taught that a thread switch control register may be implemented for each thread for holding a state of that thread and that the control state for the second thread may be stored during execution of the first thread. See column 13, lines 20-45. This control register (and control state) allows the system to specify which types of events would result in the switching of the associated thread, thereby increasing flexibility by allowing the user to choose how the thread may or may not be switched. Consequently, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Joy to include the control register of Borkenhagen as part of the thread state, where the thread state of the second thread is stored during execution of the first thread.

78. Claim 15 is rejected under 35 U.S.C. 103(a) as being unpatentable over Joy in view of McCrackin and further in view of Levy et al., U.S. Patent No. 6,314,511 (herein referred to as Levy).

79. Referring to claim 15, Joy in view of McCrackin has taught a system as described in claim 17. While Joy has taught that said first set of data storage devices comprises registers (see column 8, lines 27-44, and column 3, lines 3-10), Joy has not taught that the registers are shared by a plurality of threads. Instead, Joy has taught that each thread gets its own register file. However, Levy has taught that some tests have shown that shared registers provide performance gains when compared to dedicated per-thread register designs, as taught by Joy. In addition, by sharing registers, the total size of the register file is reduced (since you don't have to have a separate register file for each thread) without sacrificing performance. See Levy, column 8, line

65, to column 9, line 3. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Joy such that the registers are shared by the threads instead of replicated.

80. Claims 34-41 and 48-55 are rejected under 35 U.S.C. 103(a) as being unpatentable over Borkenhagen in view of Gee and further in view of Ramakrishnan.

81. Referring to claim 34, Borkenhagen in view of Gee has taught a system as described in claim 1. Borkenhagen has not taught that said hardware thread scheduler includes a thread identifier for identifying at least one hard-real-time (HRT) thread and at least one non-real-time thread and a HRT scheduler for regularly scheduling said HRT thread in available time quanta such that said HRT thread is scheduled to ensure the execution of the HRT in a predetermined time. However, Ramakrishnan has taught such a concept. See column 4, line 52, to column 5, line 3, and the abstract. Note that real-time and general (non real-time) threads are determined and that a real time thread is scheduled during the available time quanta such that it executes in predetermined time (i.e., in a preselected maximum time). In such a system, real-time threads, which perform time-critical tasks, are given priority over general threads. This is clear because the general threads execute for a minimum time and then after that, if a real time thread needs processing, then it will preempt that general thread. Furthermore, Ramakrishnan has taught that such a system prevents starvation of threads (since all threads get some time to process) and offers a greater degree of fairness in allocating processing resources to various tasks. See column 4, lines 11-14. As a result, in order to execute time-critical tasks in a non-starving and fair way, it would have been obvious to one of ordinary skill in the art at the time of the

invention to modify Borkenhagen be used in a time-critical environment and to include at least one HRT thread and an HRT scheduler, as taught by Ramakrishnan.

82. Referring to claim 35, Borkenhagen in view of Gee in view of Ramakrishnan has taught a system as described in claim 34. Ramakrishnan has further taught that said time quanta is at least one instruction cycle. See the abstract and note that real-time threads are scheduled for a preselected maximum amount of time. This time is inherently at least one cycle because if it were any less (zero cycles), then the thread would never execute.

83. Referring to claim 36, Borkenhagen in view of Gee in view of Ramakrishnan has taught a system as described in claim 34. Ramakrishnan has further taught that said hardware thread scheduler schedules a non-real-time (NRT) thread to replace a scheduled HRT thread if said HRT is idle. See Fig.3, step 74, and note that if a real-time thread is idle (has no work), then the schedule is free to replace it with another thread, where another thread is either an NRT or another HRT.

84. Referring to claim 37, Borkenhagen in view of Gee in view of Ramakrishnan has taught a system as described in claim 34. Ramakrishnan has further taught that said hardware thread scheduler schedules the execution of non-real-time (NRT) threads in quanta not allocated to HRT threads. See the Fig.2A and note that a real-time thread is allocated time and that time is the real-time thread's time (step 52). After the HRT is done, an NRT may be scheduled. That is, they are both not scheduled in the same quanta (at the same time). The HRT is to execute (assuming the HRT is first to execute), and then the NRT is to execute.

85. Referring to claim 38, Borkenhagen in view of Gee in view of Ramakrishnan has taught a system as described in claim 37. Ramakrishnan has further taught that said hardware thread

scheduler regularly schedules NRT threads to be executed. See the abstract and note that there may be a plurality of NRTs for scheduling.

86. Referring to claim 39, Borkenhagen in view of Gee in view of Ramakrishnan has taught a system as described in claim 34. Borkenhagen has further taught:

a) a first storage device for storing program instructions, said processor fetching instructions from the first storage device within a first fetch period. See Fig.1, component 150. The cache will be fetched from during the time that the instructions needed are in the cache.

b) a second storage device for storing program instructions, said processor fetching instructions from the second storage device within a second fetch period. See Fig.1, component 140. The main memory will be fetched from during the time that the instructions needed are not in the cache.

c) wherein said first fetch period is substantially shorter than said second fetch period. See column 3, lines 13-17. Fetching from a cache is shorter than fetching from main memory.

87. Referring to claim 40, Borkenhagen in view of Gee in view of Ramakrishnan has taught a system as described in claim 39. Borkenhagen has not taught that said first storage device for storing program instructions comprises a static RAM. However, Official Notice is taken that virtually all caches are implemented with static RAM (SRAM) and that SRAM and its advantages are well known and accepted in the art. SRAM is fast, which makes it suitable for caches, and unlike DRAM, it does not need to be refreshed in order to maintain its contents. Consequently, for speed and storage ability, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Borkenhagen's instruction cache such that it is implemented in SRAM.

88. Referring to claim 41, Borkenhagen in view of Gee in view of Ramakrishnan has taught a system as described in claim 39. Borkenhagen has not taught that said second storage device for storing program instructions comprises a flash memory. However, Official Notice is taken that flash-based main memories and their advantages are well known and accepted in the art. A computer hierarchy based upon volatile main memory loses all information in main memory when power is turned off. A flash-based non-volatile main memory, however, reduces or eliminates the lengthy process of obtaining information from disk when power is turned on. Therefore flash main memory based computer system has higher system performance when a program is initially executed than would a volatile main memory based computer system. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Borkenhagen's main memory such that it is a flash-based main memory.

89. Referring to claim 48, Borkenhagen in view of Gee has taught a method as described in claim 46. Borkenhagen has not taught identifying which of the said program threads said processor executes according to a hard real-time (HRT) execution schedule. However, Ramakrishnan has taught the concept of real-time threads. See column 4, line 52, to column 5, line 3, and the abstract. Note that real-time threads are identified and are those that perform time-critical work and therefore should be given priority in the system. In such a system, real-time threads are given priority over general threads (note that Borkenhagen has also taught using priority with threads in the abstract). This priority concept is clear in Ramakrishnan because the general threads execute for a minimum time and then after that, if a real time thread needs processing, then it will preempt that general thread. Furthermore, Ramakrishnan has taught that such a system prevents starvation of threads (since all threads get some time to process) and

offers a greater degree of fairness in allocating processing resources to various tasks, while allowing important work to get done. See column 4, lines 11-14. As a result, in order to execute time-critical tasks in a non-starving and fair way, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Borkenhagen to be used in a time-critical environment and to identify which of the program threads the processor executes according to an HRT execution schedule, as taught by Ramakrishnan.

90. Referring to claim 49, Borkenhagen in view of Gee in view of Ramakrishnan has taught a method as described in claim 48. Gee has further taught allocating available processing time of the processor among at least the first and second threads according to the predetermined fixed execution schedule. Please recall the rejection of claim 46 and note that time is allocated to periodic threads according to the schedule fixed by the structure of Fig.19.

91. Referring to claim 50, Borkenhagen in view of Gee in view of Ramakrishnan has taught a method as described in claim 49. Gee has further taught that the allocating comprises dividing the available execution time into a plurality of quanta, each quanta corresponding to a number of instruction cycles for execution of a thread. Again, see the rejection of claim 46. Each thread is allocated a number of cycles (quanta) in which to execute. This number is associated with the timeout value of the hardware timer.

92. Referring to claim 51, Borkenhagen in view of Gee in view of Ramakrishnan has taught a method as described in claim 50. Gee has further taught that at least one quantum corresponds to a thread that is scheduled to execute periodically after a fixed number of execution cycles. Please recall the rejection of claim 46 and Fig.19 of Gee.

93. Referring to claim 52, Borkenhagen in view of Gee in view of Ramakrishnan has taught a method as described in claim 48. Ramakrishnan has further taught that identifying further comprises identifying at least one hard real-time (HRT) thread and at least one non real-time (NRT) thread. See column 4, line 52, to column 5, line 3, and the abstract. Note that real-time threads are identified and are those that perform time-critical work and therefore should be given priority in the system over general threads, which are the less time-critical threads.

94. Referring to claim 53, Borkenhagen in view of Gee in view of Ramakrishnan has taught a method as described in claim 52. Ramakrishnan has further taught scheduling the HRT thread in available time quanta such that said HRT thread is scheduled to ensure the execution of the HRT thread within a predetermined time. See column 4, line 52, to column 5, line 3, and the abstract. Note that HRT threads are given a maximum time in which to execute. This ensures the execution of the HRT within that maximum time.

95. Referring to claim 54, Borkenhagen in view of Gee in view of Ramakrishnan has taught a method as described in claim 52. Ramakrishnan has further taught scheduling an NRT thread for a quantum allocated for an HRT thread if said HRT thread is idle. See Fig.3, step 74, and note that if a real-time thread is idle (has no work), then the schedule is free to replace it with another thread, where another thread is either an NRT or another HRT.

96. Referring to claim 55, Borkenhagen in view of Gee in view of Ramakrishnan has taught a method as described in claim 52. Ramakrishnan has further taught scheduling NRT threads in quanta not allocated for HRT threads. See the Fig.2A and note that a real-time thread is allocated time and that time is the real-time thread's time (step 52). After the HRT is done, an NRT may be scheduled. That is, they are both not scheduled in the same quanta (at the same

time). The HRT is to execute (assuming the HRT is first to execute), and then the NRT is to execute.

97. Claim 44 is rejected under 35 U.S.C. 103(a) as being unpatentable over Borkenhagen in view of Gee and further in view of Levy.

98. Referring to claim 44, Borkenhagen in view of Gee has taught a system as described in claim 1. While Borkenhagen has hinted at sharing of resources (column 5, lines 56-57), Borkenhagen has not explicitly taught that said first set of storage devices comprises registers shared by a plurality of threads. However, Levy has taught that some tests have shown that shared registers provide performance gains when compared to dedicated per-thread register designs. In addition, by sharing registers, the total size of the register file is reduced (since you don't have to have a separate register file for each thread) without sacrificing performance. See Levy, column 8, line 65, to column 9, line 3. As a result, it would have been obvious to one of ordinary skill in the art at the time of the invention to modify Borkenhagen such that the registers are shared by the threads.

Response to Arguments

99. Applicant's arguments filed on September 17, 2008, have been fully considered but they are not persuasive.

100. Applicant argues the novelty/rejection of claim 1 on pages 18-20 of the remarks, in substance that:

> (1) "In contrast, Gee discloses using a single Java embedded microprocessor (JEM) to execute multiple Java Virtual Machines (JVMs). Each JVM is assigned a fixed area of memory and allotted a fixed amount of time in which to operate. After the fixed amount of time, the JEM

invokes a master JVM to handle system duties and uses the master JVM to subsequently invoke another JVM. Gee, col. 3, lines 50-65. To switch between virtual machines, the JEM executes thread control blocks (TCBs) and executive control blocks (ECBs), which are both constructed to look like JAVA objects to simplify manipulation by JAVA software. Gee, col. 19, lines 40-59. **Hence, the JVM does not use a hardware thread scheduler, but uses software processes (the TCBs and ECBs) to perform context switching between virtual machines.** As software, the ECBs of Gee must be executed by the JEM in order to specify a thread switching schedule. Because of this, the ECBs and TCBs of Gee execute at too high a level to allocate specific numbers of instruction cycles, as claimed."

(2) "Gee discloses that execution of the ECB causes "interstitial" activity while switching between threads, which consumes a number of cycles, as illustrated in FIG. 14. Gee, col. 23, line 65 to col. 24, line 13; FIG. 14. This "interstitial" activity performs various system-level functions for the JEM, so that the activity does not perform for a defined number of cycles, causing imprecise context switching which renders Gees incompatible with cycle-specific scheduling. For example, an ECB indicating that an application should be executed every 100 time units, for example, could not indicate a specific number of cycles, because any varying number of cycles could be lost while performing the "interstitial" activity necessary to change contexts and subsequently switching to a new thread."

101. These arguments are not found persuasive for the following reasons:

a) As bolded above, applicant's first argument states that Gee uses a hardware scheduler instead of a hardware scheduler. While a master JVM is used in the scheduling process, hardware is also used in the form of times (see the abstract and Fig.18A and 18B) and in the piano roll table, which is clearly memory (i.e., hardware) based. Therefore, a hardware scheduler is used.

b) Regarding the 2nd argument, applicant is correct in pointing out that interstices exist between executing contexts. However, this does not result in imprecise scheduling, nor does it preclude from scheduling in the manner claimed by applicant. Specifically, see the abstract, Figs.18A and 18B, and claims 1-2 of Gee. Note that when one thread ends, the master JVM is initiated, and the amount of time the master JVM operates is limited by a timer. Once the timer expires, the next thread's proxy thread is activated (see claims 1-2 of Gee). However, from Fig.18A, this proxy thread's execution is considered part of the next thread's overall execution. Therefore, Gee has taught "thread-switching at a fixed time according to a predetermined fixed schedule, said schedule specifying that the first thread should be allocated processing time every first

number of cycles and that the second thread should be allocated processing time every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles," as claimed. The following breakdown should help clarify. In the rejection of claim 1 above, the examiner gave an example in which the following thread sequence would occur: t21-t19-t21-t17-t21-t19-t21-t17. As shown in Fig.14 and 18A, each thread (JVM) includes a proxy thread (in this case, we'll call them p21, p19, and p17, respectively), which is considered to be part of its respective thread (Fig.18B). Therefore, after p21 and t21 execute for their fixed time slice, a fixed, timer-based interstice exists in which the master operates, and then p19 and t19 follow. Following this rule, the following pattern of execution is realized with the fixed interstice being denoted as "fi".

p21-t21-fi-p19-t19-fi-p21-t21-fi-p17-t17-fi-p21-t21-fi-p19-t19-fi-p21-t21-fi-p17-t17

However, recall from Fig.18B that the proxy thread is actually part of the overall thread because the proxy thread's execution is considered to be part of the overall thread's time slice. Hence, any px-tx combination above may simply be referred to as "sx", which is a time slice amount associated with thread x. Consequently, the above schedule can be simplified as:

s21-fi-s19-fi-s21-fi-s17-fi-s21-fi-s19-fi-s21-fi-s17

Now, it can be seen that this schedule reads on the claimed schedule because thread switching occurs at a fixed time according to a predetermined fixed piano roll schedule. For simplicity, assume that each sx takes 10 cycles and each fi takes 5 cycles. Recall that all fi's are equal because they are based on a timer. The sx time slices could also differ amongst each other,

but whether they're the same or different is a non-issue because both scenarios read on the claimed invention. With the above assumptions, s21 will be allocated processing time every 30 cycles and s19 is allocated time every 60 cycles.

102. Applicant's arguments with respect to claim 17 on pages 22-26 of the remarks have been considered but are moot in view of the new ground(s) of rejection.

103. Applicant's remaining arguments that the deficiencies of the main references are not taught by secondary references are moot because such deficiencies do not exist in view of the new grounds of rejection.

Conclusion

104. Applicant's amendment necessitated the new ground(s) of rejection presented in this Office action. Accordingly, **THIS ACTION IS MADE FINAL**. See MPEP § 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event,

however, will the statutory period for reply expire later than SIX MONTHS from the date of this final action.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to DAVID J. HUISMAN whose telephone number is (571)272-4168. The examiner can normally be reached on Monday-Friday (8:00-4:30).

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Eddie Chan can be reached on (571) 272-4162. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/David J. Huisman/
Primary Examiner, Art Unit 2183
November 3, 2008